Laboratorul 4

Ce reprezinta sincronizarea?

Tinand cont ca entitatea prin care se executa cod este firul de executie (si nu procesul), sincronizarea se realizeaza la nivel de fir de executie. Chiar si atunci cand vorbim de sincronizare intre procese, ne referim de fapt la sincronizare intre firele de executie ale proceselor

Sincronizarea este necesara atunci cand sunt accesate resure comune si exista cel putin un fir de executie care modifica resursa. Sincronizarea reprezinta abilitatea firelor de executie ce functioneaza in mod concurent de a-si coordona activitatea prin schimb de informatie. In Windows Api, se realizeaza prin excludere mutuala.

Excluderea mutuala reprezinta modalitatea prin care mai multe unitati de executie pot accesa date comune in asa fel incat doar o unitate are acces la date la un moment de tip. Acea sincronizare in care firele de executie au acces la resursa comuna in mod alternativ reprezinta un caz particular de sincronizare.

Mecanisme de sincronizare?

1. **Sectiunea Critica**

Reprezinta cel mai rapid mecanism de sincronizare, dar poate fi folosit doar pentru a sincroniza firele de executie din acelasi proces. Pentru a putea fi folosita, trebuie ca in cadrul procesului sa fie initializat un obiect(structura) de tip CRITICAL_SECTION. Fiecare fir de executie va concura pentru sectiunea critica, dar doar unul singur va reusi, restul urmand sa intre in starea wait. Cand firul care a avut acces la sectiune critica renunta la obiect, alt fir din cele blocate va avea acces.

Pasii sunt urmatorii:

1. Declarare obiect de tip sectiune critica
```
CRITICAL_SECTION CriticalSection
```
2. Unul din fire (deobicei cel princiapl) initializeaza sectiunea critica:
```
InitializeCriticalSection(&CriticalSection)
```
3. Zona de cod care acceseaza reusrsele coune vor fi marcate cu:
```
EnterCriticalSection(&criticalSection);
      ZONA DE COD CARE ACCESEAZA resursa comuna
LeaveCriticalSection(&criticalSection);
```
4. Pentru dealocare resurse se foloseste functia
```
DeleteCriticalSection(&CriticalSection)
```

2. **Mutex**

Asemanator ca functionare cu sectiunea critica, dar poate fi folosit si pentru sincronizarea firelor de executie din alte procese. Fiecare fir incearca sa obtina obiecutl Mutex, dar dar unul va reusi, restul urmand sa intre in starea wait. Cine detine mutexul, va accesa zona de date comune. Cand se elibereaza mutexul, alt fir va fi deblocat.

Pasii sunt urmatorii:

1. Un fir de executie creaza mutextul

```
hMutex = CreateMutex(...,numeMutex)


HANDLE WINAPI CreateMutex(
  _In_opt_  LPSECURITY_ATTRIBUTES lpMutexAttributes,
  _In_      BOOL bInitialOwner,
  _In_opt_  LPCTSTR lpName);
```

lpMutexAttributes	Drepturile de securitate pentru noul obiect de tip Mutex.poate fi NULL pentru drepturi standard
bInitialOwner	Daca se doreste ca firul care creaza mutexul sa-l detina de la inceput
lpName	Numele obiectului, daca se doreste sa poata fi gasit de alte procese

2. Celelate fire de executie (daca nu sunt din acelasi proces) trebuie sa obtina accela mutex

```
hMutex = OpenMutex(...,numeMutex)

HANDLE WINAPI OpenMutex(
        _In_  DWORD dwDesiredAccess,
        _In_  BOOL bInheritHandle,
        _In_  LPCTSTR lpName
);
```

dwDesiredAccess	Ce actiuni se doresc asupra obiectului. Deobicei, SYNCHRONIZE
bInheritHandle	Daca acest handle poate fi mostenit de un process copil
lpName	Numele mutexului (acelasi care a fost dat la CrateMutex, dar din alt proces)

3. Fiecare fir incearca sa detina mutex-ul, prin apelul functiei WaitForSingleObject. Daca-l obtine, va accesa zona de cod care acceseaza resursa comuna

```
WaitForSingleObject(hMutex, INFINITE);
      ZONA DE COD CARE ACCESEAZA resursa comuna
ReleaseMutex(hMutex);
```

```
DWORD WINAPI WaitForSingleObject(
  _In_  HANDLE hHandle,
  _In_  DWORD dwMilliseconds
);
```

Functia WaitForSingleObject suspenda executia firului curent pana apare un eveniment. Evenimentul este specific obiectului pe care-l astepata si este indicat prin primul parametru, hHandle. Daca primul parametru este un handle catre un Mutex, va astepta ca mutexul sa fie eliberat. Daca este spre un proces, va astepta ca procesul sa se termine, daca este un fir, va astepta ca firul sa se termine. Etc.

hHandle	Handle catre obiectul pe care se doreste ca firul sa-l astepte.
dwMiliseconds	Numarul maxim de secunde care sa astepte. Daca se speciica INFINITE, va astepta pana cand obiectul este eliberat

4. Pentru dealocare, fiecare proces va inchide handle-ul catre mutex
   `CloseHandle(hMutex)`
   Dealocarea propriu-zisa se va face atunci cand nu mai exista nici o referinta catre mutex

3. Semafor

Semaforul este folosit atunci cand avem mai multe fire de executie ce acceseaza o resursa comuna, dar dorim ca maxim un anumit numar de fire sa aiba acces spre acea resursa. Daca acel maxim este 1, atunci avem de fapt un Mutex. Un exemplu in care ar putea fi util semaforul este urmatorul. Avem 2 imprimante conectate la un servicu de imprimare. Toate procesele care doresc sa printeze ceva, vor accesa acel serviciul. Tinand cont ca numarul de imprimante este 2, serviciu va trebui sa limiteze numarul maxim de procese ce pot accesa imprimantele, la 2.

Fiecare fir incearca sa obtina obiectul semafor. De fiecare data cand un fir obtine accesul, un contor este decrementat pana ajunge la 0. Cat timp contorul este 0, orice fir ce acceseaza semaforul va intra in starea wait. Cand un fir elibereaza semaforul, contorul creste cu 1.

Pasii sunt urmatorii:

1. Un fir de executie creaza semaforu;
   `hSemaphore = CreateSemaphore(...,numeSemafor)`

```
HANDLE WINAPI CreateSemaphore(
  _In_opt_  LPSECURITY_ATTRIBUTES lpSemaphoreAttributes,
  _In_      LONG lInitialCount,
  _In_      LONG lMaximumCount,
  _In_opt_  LPCTSTR lpName
);
```

lpSemaphoreAttributes	Drepturile de securitate pentru noul obiect de tip Semaphore.poate fi NULL pentru drepturi standard
lInitialCount	Numarul initial de fire care au deja accesa la resursa. Deobicei, 0
lMaximumCount	Numarul maxim de fire care pot accesa resursa comuna la un moment dat
lpName	Numele obiectului, daca se doreste sa poata fi gasit de alte procese

2. Celelate fire de executie (daca nu sunt din acelasi proces) trebuie sa gaseasca obiectul semaphore pentru a putea lucra cu el

```
hSemaphore = OpenSemaphore(...,numeSemafor)

HANDLE WINAPI OpenSemaphore(
  _In_  DWORD dwDesiredAccess,
  _In_  BOOL bInheritHandle,
  _In_  LPCTSTR lpName
);
```

dwDesiredAccess	Ce actiuni se doresc asupra obiectului. Deobicei, SEMAPHORE_MODIFY_STATE
bInheritHandle	Daca acest handle poate fi mostenit de un process copil
lpName	Numele semaforului (acelasi care a fost dat la CrateSemaphore, dar din alt proces)

3. Fiecare fir incearca sa detina semaforul, prin apelul functiei WaitForSingleObject. Daca-l obtine, va accesa zona de cod care acceseaza resursa comuna

```
WaitForSingleObject(hSemaphore, INFINITE);
       ZONA DE COD CARE ACCESEAZA resursa comuna
ReleaseSemaphore(hSemaphore,...);

BOOL WINAPI ReleaseSemaphore(
  _In_       HANDLE hSemaphore,
  _In_       LONG lReleaseCount,
  _Out_opt_  LPLONG lpPreviousCount
);
```

hSemaphore	Handle-ul catre obiectul de tip semafor
lReleaseCount	Cu cat sa fie incrementat numarul de fire ce pot avea acces la semafor. Deobicei este 1
lpPreviousCount	Daca se doreste, se functia poate returna numarul precedent de fire care a accesat acea resursa.

4. Pentru dealocare, fiecare proces va inchide handle-ul catre semafor

```
CloseHandle(hSemaphore)
```

Dealocarea propriu-zisa se va face atunci cand nu mai exista nici o referinta catre semafor

4. **Event**

   Evenimentul reprezinta un obiect cu doua stari: setat/nesetat. Poate fi folosit pentru sincronizarea irelor de executie din orice proces.

   Unul sau mai multe fire asteapta sa se semnalizeze un eveniment de catre alt fir. De exemplu: un fir a terminat de scris datele, iar alte fire care asteapta pot incepe sa le citeasca.

1. Un fir de executie creaza evenimentul

```
hEvent  = CreateEvent (...,numeEvent)


HANDLE WINAPI CreateEvent(
  _In_opt_  LPSECURITY_ATTRIBUTES lpEventAttributes,
  _In_      BOOL bManualReset,
  _In_      BOOL bInitialState,
  _In_opt_  LPCTSTR lpName
);
```

lpEventAttributes	Drepturile de securitate pentru noul obiect de tip event.poate fi NULL pentru drepturi standard
bManualReset	Specifica daca dupa ce un fir de executie isi reporneste executia dupa ce un eveniment este semnalizat, evenimentul va fi resetat
binitialState	Daca este setat initial evenimentul
lpName	Numele obiectului, daca se doreste sa poata fi gasit de alte procese

2. Celelate fire de executie (daca nu sunt din acelasi proces) trebuie sa gaseasca obiectul eveniment pentru a putea lucra cu el

```
hEvent= OpenEvent(...,numeEvent)

HANDLE WINAPI OpenEvent(
  _In_  DWORD dwDesiredAccess,
  _In_  BOOL bInheritHandle,
  _In_  LPCTSTR lpName
);
```

dwDesiredAccess	Ce actiuni se doresc asupra obiectului. Deobicei, EVENT_MODIFY_STATE
bInheritHandle	Daca acest handle poate fi mostenit de un process copil
lpName	Numele evenimentului (acelasi care a fost dat la CreateEvent, dar din alt proces)

3. Fiecare fir va astepta semnala evenimentului prin apelul functiei WaitForSingleObject. Daca evenimentul nu este setat, atunci firele intra in starea waiting. Atunci cand se seteaza evenimentul, unul (daca bManualReset este setat pe false) sau mai multe (daca bManualReset este setat pe true) isi vor incepe executia

```
WaitForSingleObject(hEvent, INFINITE);
      ZONA DE COD CARE ACCESEAZA resursa comuna
```

4. Un fir va trebui sa seteze evenimentul, pentru a semnaliza celelalte fire ca-si pot continua executia. Acest lucru se face prin functia SetEvent

```
SetEvent(hEvent)
```

5. Pentru dealocare, fiecare proces va inchide handle-ul catre event
   `CloseHandle(hEvent)`
   Dealocarea propriu-zisa se va face atunci cand nu mai exista nici o referinta catre semafor

5. **Timer**

   Timer-ul reprezinta un eveniment care va fi setat la un moment stabilit inainte. Poate fi folosit pentru semnalizarea firelor de executie din orice proces.
   Poate functiona atat ca un eveniment ce va fi semnalizat periodic cat si ca un eveniment ce va fi semnalizat o singura data.
   Poate fi programat sa functioneze ca obiect de sincronizare (va fi dezactivat dupa ce un fir care asteapta pe el a iesit din starea wait), cat si ca un eveniment ce trebuie resetat manual (va ramane semnalizat pana cand se reseteaza).

1. Un fir de executie creaza timer-ul

```
hEvent  = CreateWaitableTimer (...,numeTimer)

HANDLE WINAPI CreateWaitableTimer(
  _In_opt_  LPSECURITY_ATTRIBUTES lpTimerAttributes,
  _In_      BOOL bManualReset,
  _In_opt_  LPCTSTR lpTimerName
);
```

lpTimerAttributes	Drepturile de securitate pentru noul obiect de tip timer.poate fi NULL pentru drepturi standard
bManualReset	Specifica daca dupa semnalizare se va reseta autoamt, sau trebuie resetat programmatic
lpName	Numele obiectului, daca se doreste sa poata fi gasit de alte procese

2. Un fir de executie seteaza timer-ul (deobicei, acelasi care l-a creat) prin apelul functiei SetWaitableTimer

```
BOOL WINAPI SetWaitableTimer(
  _In_      HANDLE hTimer,
  _In_      const LARGE_INTEGER *pDueTime,
  _In_      LONG lPeriod,
  _In_opt_  PTIMERAPCROUTINE pfnCompletionRoutine,
  _In_opt_  LPVOID lpArgToCompletionRoutine,
  _In_      BOOL fResume
);
```

hTimer	Handle catre obiectul de tip Timer.
pDueTime	Momentul la care va fi semnalizat ceasul prima data, in multiplu de 100 nanosecunde. Daca valoarea este pozitiva, atunci este timp absolut, daca este negative atunci este timp relativ
lPeriod	Daca se doreste ca ceasul sa fie periodic, atunci se specfica perioada(in milisecunde)
pfnCompletionRoutine	O functie ce poate fi apelata atunci cand se seteaza timer-ul Functia trebuie sa fie de urmatorul tip: `VOID (APIENTRY *PTIMERAPCROUTINE)( _In_opt_ LPVOID lpArgToCompletionRoutine, _In_ DWORD dwTimerLowValue, _In_ DWORD dwTimerHighValue );`
lpArgToCompletionRoutine	Argumentul ce poate fi transmis functiei anterioare
fresume	Specifica daca sistmel sa fie activat in caz ca este in „suspended power conservation mod”

3. Celelate fire de executie (daca nu sunt din acelasi proces) trebuie sa gaseasca obiectul timer pentru a putea lucra cu el

```
hEvent= OpenWaitableTimer(...,numeEvent)

HANDLE WINAPI OpenWaitableTimer(
  _In_  DWORD dwDesiredAccess,
  _In_  BOOL bInheritHandle,
  _In_  LPCTSTR lpTimerName
);
```

dwDesiredAccess	Ce actiuni se doresc asupra obiectului. Deobicei, TIMER_MODIFY_STATE
bInheritHandle	Daca acest handle poate fi mostenit de un process copil
lpName	Numele timer-ului(acelasi care a fost dat la CreateWaitableTimer, dar din alt proces)

4. Fiecare fir va astepta semnala timer-ului prin apelul functiei WaitForSingleObject. Atunci cand se seteaza evenimentul, unul (daca bManualReset este setat pe false) sau mai multe (daca bManualReset este setat pe true) isi vor incepe executia

```
WaitForSingleObject(hEvent, INFINITE);
        ZONA DE COD CARE ACCESEAZA resursa comuna
```

   In caz ca la functia SetWaitableTimer a fost seatata o functie ca parametru, atunci aceasta va fi executata doar daca firul este pus intr-o stare alerta. Pentru a intra in starea alerta, se apeleaza functia SleepEx(`INFINITE`, true)

5. Pentru dezactivare ceas, se apeleaza fucntia:
   `CancelWaitableTimer(hTimer)`

6. Pentru dealocare, fiecare proces va inchide handle-ul catre timer
   `CloseHandle(hTimer)`
   Dealocarea propriu-zisa se va face atunci cand nu mai exista nici o referinta catre timer.

Exemple:

Sectiune Critica

Mutex

Semafor

Event

Timer